

**Create a Docker Image****Step 1: Update the Code**

1. Add dockerfile as below to HelloWorldApp.Web

Filename: **HelloWorldApp.Web\Dockfile**

```
FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /src
COPY ["HelloWorldApp.Web/HelloWorldApp.Web.csproj", "HelloWorldApp.Web/"]
RUN dotnet restore "HelloWorldApp.Web/HelloWorldApp.Web.csproj"
COPY . .
WORKDIR "/src/HelloWorldApp.Web"
RUN dotnet build "HelloWorldApp.Web.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "HelloWorldApp.Web.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "HelloWorldApp.Web.dll"]
```

**Azure Pipeline for Build and Publish Docker Image**

Azure Pipelines can be used to build images for any repository containing a Dockerfile. Building of both Linux and Windows containers is possible based on the agent platform used for the build.

**Create a New Service Connection**

Organization Properties → Service Connection → **New Service Connection** → **Docker Registry** → Select **Docker Hub**

**Create a New Pipeline**

1. Project → Pipelines → New Pipeline → Azure Repos → Docker (Build and push an image to Azure container registry)

**Azure-pipeline.yml**

```
trigger:
- none

stages:
- stage: Build
  displayName: Build and push stage
  jobs:
  - job: Build
    displayName: Build
    pool:
      vmImage: 'ubuntu-latest'
    steps:
    - task: Docker@2
      displayName: Build and push an image to container registry
      inputs:
        command: buildAndPush
        containerRegistry: 'Docker Connection To Azure'
        dockerfile: '**/Dockerfile'
        buildContext: .
        repository: 'sandeepsoni/helloworldapp.web' #sandeepsoni here is login id in Docker Hub and remove
that for Azure Container Registry
        tags: '$(Build.BuildId)'
```

**Classic Pipeline:**

Pipeline  
Build pipeline

Get sources  
HelloWorldApp1 master

Agent job 1  
Run on agent

Build an image  
Docker

Task version 2.\*

Display name \*  
Build an image

Container Repository ^

Container registry (i) | Manage

Docker Hub Connection

Container repository (i)  
sandeepsoni/helloworldapp

Commands ^

Command \* (i)  
buildAndPush

Dockerfile \* (i)  
\*\*/Dockerfile

Build context (i)  
.

Tags (i)  
\$(Build.BuildId)

For Build Context, the default value is "\*\*" (uses the location of Dockerfile as Build Context).

### Azure Container Registry

- **Azure Container is a private registry allows you to store and manage docker container images**
- Use container registries in Azure with your existing container development and deployment pipelines.

#### Use Azure Container Registry to:

1. Store and manage container images across all types of Azure deployments
2. Use familiar, open-source Docker command line interface (CLI) tools
3. Keep container images near deployments to reduce latency and costs
4. Simplify registry access management with Azure Active Directory
5. Maintain Windows and Linux container images in a single Docker registry

#### Create Container Registry Using Portal

1. **Create a resource → Containers → Azure Container Registry.**
2. Under **Admin user**, select **Enable**. Take note of the following values:

- Login server
- Username
- password

### 3. Login to ACR

```
docker login --username dssdemo --password X3bl/uVbrNJ8lgfLXqjDV4zQVWRJgOI1  
dssdemo.azurecr.io
```

### 4. Tag Local Images

```
docker image tag sandeepsoni/hellowebapp:v1 dssdemo.azurecr.io/hellowebapp:v1
```

### 5. Push images to ACR

```
docker push dssdemo.azurecr.io/hellowebapp:v1
```

## Create a Service Connection

1. Azure Portal → Select Container Registry → Access Keys → Copy Username, Password and Docker Server
2. Azure DevOps Project → Properties → Service Connection → Docker Registry → Enter details captured in prev step as below

Registry type

☐ Docker Hub ☒ Others ☐ Azure Container Registry

Docker Registry

https://dsdemoregistry.azurecr.io

Docker ID

dsdemoregistry

Docker Password

\*\*\*\*\*

Email (optional)

Details

Service connection name

Azure Container Registry Connection

## Azure-pipeline.yml

```
trigger:  
- none  
  
pool:  
  vmImage: ubuntu-latest  
  
steps:
```

```
- task: Docker@2
  inputs:
    containerRegistry: 'Azure Container Registry Connection'
    repository: 'helloworldapp'
    command: 'buildAndPush'
    Dockerfile: '**/Dockerfile'
    buildContext: '.'
```

### Deploying to Web App

You can automatically deploy your application to an Azure Web App for Linux Containers after every successful build.

You must supply an Azure service connection to the AzureWebAppContainer task. Add the following YAML snippet to your existing **azure-pipelines.yaml** file. Make sure you add the service connection details in the variables section as shown below-

```
trigger:
- master

resources:
- repo: self

variables:
  tag: '$(Build.BuildId)'

stages:
- stage: Build
  displayName: Build image
  jobs:
  - job: Build
    displayName: Build
    pool:
      vmImage: ubuntu-latest
    steps:
    - task: Docker@2
      displayName: Build an image
      inputs:
        repository: 'sandeepsoni/helloworldapp'
        command: 'build'
        Dockerfile: '**/dockerfile'
```

```
buildContext: '.'
tags: '${tag}'

- task: Docker@2
  displayName: Build an image
  inputs:
    containerRegistry: 'Docker Hub Connection'
    repository: 'sandeepsoni/helloworldapp'
    command: 'push'
    tags: '${tag}'

- stage: Deploy
  displayName: Deploy image
  jobs:
    - job: Deploy
      displayName: Deploy
      pool:
        vmImage: ubuntu-latest
      steps:
        - task: AzureWebAppContainer@1
          inputs:
            azureSubscription: 'Azure Training – SS2'
            appName: 'dsdockerappservicedemo'
            containers: 'dsdemoregistry.azurecr.io/helloworldapp:${Build.BuildId}'
```

OR

```
- task: AzureRmWebAppDeployment@4
  inputs:
    ConnectionType: 'AzureRM'
    azureSubscription: 'Azure Connection - SS2'
    appType: 'webAppContainer'
    WebAppName: 'dshelloworldapp'
    DockerNamespace: 'dsdemoregistry.azurecr.io'
    DockerRepository: 'helloworldapp'
    DockerImageTag: '${Build.BuildId}'
```

**AppSettings:** '-**DOCKER\_REGISTRY\_SERVER\_USERNAME** dssdemo123 -  
**DOCKER\_REGISTRY\_SERVER\_PASSWORD** B7Vo1f6hBxNdUA8sWwcScBdcLTsMEE7='

**Note:** The last line AppSettings is NOT REQUIRED if the username and password are directly provided in Azure App Service → Configuration → AppSettings.

#### Release Pipeline:

The screenshot shows the configuration for a release pipeline task named 'Run on agent'. The task is 'Azure App Service Deploy: dssdemoapp123-con'. The configuration details on the right are as follows:

- Display name:** Azure App Service Deploy: dssdemoapp123-con
- Connection type:** Azure Resource Manager
- Azure subscription:** Azure Connection - SS2
- App Service type:** Web App for Containers (Linux)
- App Service name:** dssdemoapp123-con
- Registry or Namespace:** dssdemo123.azurecr.io
- Image:** helloworldapp
- Tag:** \$(Build.BuildId)
- Startup command:** -DOCKER\_REGISTRY\_SERVER\_URL dssdemo123.azurecr.io -DOCKER\_REGISTRY\_SERVER\_USERNAME dssdemo123 -DOCKER\_REGISTRY\_SERVER\_PASSWORD B7Vo1f6hBxNdUA8sWwcScBdcLTsMEE7=

#### Very Important:

The above task is only setting the value of AppService - AppSettings key: DOCKER\_CUSTOM\_IMAGE\_NAME = <DockerNamespace>/<DockerRepository>:DockerImageTag

The Appservice is then responsible for pulling the image from the Registry using the U/P as mentioned in its following appsettings:

- **DOCKER\_REGISTRY\_SERVER\_URL**
- **DOCKER\_REGISTRY\_SERVER\_USERNAME**
- **DOCKER\_REGISTRY\_SERVER\_PASSWORD**

**Note:** Azure Pipeline doesn't push image to App Service, it only sets the properties...

#### Note the App Service Settings after the Pipeline is executed:

```
[
{
  "name": "DOCKER_CUSTOM_IMAGE_NAME",
  "value": "dssdemo123.azurecr.io/helloworldapp:150",
  "slotSetting": false
},
{
  "name": "DOCKER_REGISTRY_SERVER_URL",
```

```
"value": "https://mcr.microsoft.com",
"slotSetting": false
},
{
  "name": "DOCKER_REGISTRY_SERVER_USERNAME",
  "value": "",
  "slotSetting": false
},
{
  "name": "DOCKER_REGISTRY_SERVER_PASSWORD",
  "value": "",
  "slotSetting": false
}
]
```

#### Running in Any VM or Local Machine using Docker Compose

- A command-line tool and YAML file format with metadata for defining and running multi-container applications.
- You define a single application based on multiple images with one or more .yaml files that can override values depending on the environment.
- After you have created the definitions, you can deploy the entire multi-container application by using a single command (docker-compose up) that creates a container per image on the Docker host.

##### 1. Create a File: docker-compose.yml

```
version: '3.4'
services:
  hellowebapp:
    image: sandeepsoni/helloworldapp
    ports:
      - '8080:80'
```

##### 2. Execute the following command to execute the application

```
docker-compose up
```